HPSS/DFS: Integration of a Distributed File System with a Mass Storage System

Rajesh Agarwalla, Madhu Chetuparambil, Craig Everhart, T. N. Niranjan
Transarc Corporation
The Gulf Tower
707 Grant Street
Pittsburgh, Pennsylvania 15219
{rajesh, madhuc, cfe, niranjan}@transarc.com
Tel: +1-412-338-4467
Fax: +1-412-338-4404

Rena Haynes, Hilary Jones
Sandia National Laboratories
P.O. Box 5800, MS0807
Albuquerque, New Mexico 87185
Sandia National Laboratories
P.O. Box 969, MS9011
Livermore, California 94551
rahayne@sandia.gov, hilary@ca.sandia.gov
Tel: +1-505-844-9149,+1-510-294-2892
Fax: +1-505-844-2067,+1-505-294-1225

Donna Mecozzi
Lawrence Livermore National Laboratory
7000 East Avenue
P.O. Box 808
Livermore, California 94551
dmecozzi@llnl.gov
Tel: +1-510-442-6020
Fax: +1-510-423-8715

Bart Parliman
Los Alamos National Laboratory
MS-B269, CIC-11
Los Alamos, New Mexico 87545
bartp@lanl.gov
Tel: +1-505-667-8410
Fax: +1-505-665-6333

Jean E. Pehkonen
IBM Software Group, Software Solutions Division
11400 Burnet Road
Internal Zip 9151
Austin, Texas 78758
jean@austin.ibm.com
Tel: +1-512-838-3457
Fax: +1-512-838-0156

Richard Ruef, Benny Wilbanks
IBM Global Government Industries
1810 Space Park Drive
Houston, Texas 77058
ruef1@llnl.gov, benny@clearlake.ibm.com
Tel: +1-510-422-0256, +1-281-335-4040
Fax: +1-281-335-4231

Vicky White
Oak Ridge National Laboratory
P.O. Box 2008
Oak Ridge, Tennessee 37831-6203
vyw@ornl.gov
Tel: +1-423-574-7999
Fax: +1-423-546-9150

Abstract: To provide cost-effective access to data in storage-intensive computing environments, mass storage systems must be integrated with underlying file systems. Previous projects have presented specialized client interfaces or have integrated mass storage with local file systems. Our approach is to integrate the distributed file system DFS, with support for DMAPI, and HPSS, a high performance mass storage system. The result is a mass storage system that includes a fully distributed file system, data transfer rates that scale to the gigabyte/sec range, and archival storage, scalable to multiple petabytes.

Introduction

Enterprises with large or networked computing environments often employ distributed file systems, providing several advantages: data consistency guarantees, location transparency, uniform name space, ease of administration, performance, and cost.

In recent years, the need to store high-resolution images, scientific data, etc., has created a serious imbalance between I/O and storage system performance and functionality[1]. To solve this imbalance, the performance and capacity of current mass storage systems must improve by orders of magnitude. Coupling mass storage systems with underlying distributed file systems, providing the best features of each and providing a seamless view of the file system, is desirable. Three emerging technologies have made this possible: The Distributed File System (DFSä)[2], the High Performance Storage System (HPSS)[3], and the Data Management Application Interface (DMAPI)[4].

In this paper, we discuss the work of the Transarc Corporation and the HPSS Collaboration to integrate DFS with HPSS using the DMAPI. We discuss past approaches to mass storage integration, the features and requirements of our system, and the overall architecture. We also describe the extensions needed in DFS, HPSS and DMAPI to support this integration, HPSS/DFS configuration options, with their corresponding performance data, and future work.

Background

Integrating mass storage with file systems to provide hierarchical storage management (HSM) has typically followed one of two philosophies:

- The file system back-end approach uses mass storage to extend the storage on the local platform.
- The mass storage file system approach implements a file system within the mass storage application.

File system back-ended mass storage allows seamless integration of mass storage with the file system platform. Leveraging the functionality of the platform simplifies HSM functionality but limits file and storage features to those supported by the local file system. These implementations require specialized software in the platform operating system, and network file services supported by the platform must be used for network access to files. Implementations based on this scheme include Data Migration Facility (DMF)[5], AMASSä [6], and E-MASSä [7].

Mass storage file system approaches typically present their file systems through specialized client interfaces. These approaches do not require operating system kernel modifications, and additional functionality is implemented directly in the HSM. This approach offers flexibility in file and storage features, such as support for network-attached peripherals, third-party transfers, and parallel I/O. However, specialized client code may be required to take advantage of any extended features. Implementations based on this scheme include Common File System (CFS)[8], UniTreeä [9], and HPSS.

Features and Requirements

The High Performance Storage System (HPSS) was developed by a collaboration of IBM and four Department of Energy laboratories: The Lawrence Livermore National Laboratory, the Los Alamos National Laboratory, Oak Ridge National Laboratory, and Sandia National Laboratories. A goal of the project was to provide mass storage capacity scalable to multiple petabytes, using standard languages and communications without operating system kernel modifications.

HPSS supports a high-speed, parallel interface[10] to achieve data transfer rates greater than 1 GB/sec for a single file. It supports third party I/O transfers[11] which allows data to move directly from the storage media to the client without intermediate server processing. HPSS supports secure access to files and directories through DCE security features, such as Access Control Lists (ACLs) and authenticated Remote Procedure Calls (RPCs). File sizes up to 2^{64} bytes are supported.

A secure, platform-independent global name space is desired for several HPSS deployment sites. DFS emerged as the primary candidate to provide this service. It has since become a requirement for HPSS to support DFS. However, the vast majority of these HPSS sites still want the high-speed I/O performance for very large files, which is the hallmark of HPSS.

DFS, developed by Transarc Corporation through the Open Software Foundation (now called The Open Group), is a highly scalable distributed file system. DFS provides a uniform view of file data to all users through *a global name space*. In addition to directory hierarchies, DFS supports logical collections of files called filesets. A fileset is a directory subtree, administered as a unit, that can be mounted in the global name space. Multiple filesets may reside on an *aggregate*, which is analogous to a disk partition. Filesets may be moved between aggregates, either on the same or different servers to achieve load balancing. DFS also supports ACLs on both directories and files to allow granting different permissions to many users and groups accessing the object. DFS uses the DCE concept of cells, and allows data access and authorization between clients and servers in different cells.

DFS uses the Episodeä (DCE LFS)[12] physical file system. This log-based file provides features like filesets, cloning (on-line backup through a fast snapshot capture mechanism), ACLs, etc., that DFS utilizes. DFS can also use other native file systems, such as UFS.

Coupling DFS with HPSS reduces the cost of storage by allowing data to be exported to media other than the disks directly attached to the DFS file server and supports files greater than the size of the disk space managed by DFS. However, the integration had to meet requirements from both Transarc and the HPSS Collaboration:

- A standard interface for coupling DFS with HPSS must be employed.
- Transparent, automatic archiving and caching of data must be provided.
- Both DFS and HPSS must support partially-resident file data.
- Changes made to a file or directory through the DFS interface must be visible through HPSS interfaces and vice versa.
- HPSS high speed file transfer mechanisms must continue to provide single file transfer rates over 1 GB/sec.

- The implementation must not effect the data transfer speeds of any data resident on disks managed by DFS, and file accesses made through DFS must not impact HPSS performance.

Fortunately, standards for interfacing file systems and data management applications (DMAP) are emerging. In the mid-nineties, the Data Management Interfaces Group (DMIG), with representatives from several file system and mass storage vendors, defined a data management interface (DMAPI) for UNIX based file systems. DMAPI, now called XDSM [13], has been adopted as a standard by the Open Group. It defines an API that can be implemented by file system vendors and used by a DMAP. The primary advantage of this standard is that mass storage vendors implementing a DMAP need not modify the OS kernel, thus enhancing portability and the ease of developing such applications.

DMAPI is a low-level interface to the physical file system. It provides the DMAP with the ability to store important attribute information with a file and allows for the generation of notifications to the DMAP on occurrence of various file system operations. DMAPI enables the DMAP to control disk storage by allowing the DMAP to move disk-resident file data to tertiary storage systems and vice-versa.

Configuration Options

A straightforward design approach satisfying both DFS and HPSS requirements was to use DMAPI to integrate DFS with HPSS. As noted, a primary objective of this effort was to provide the distributed services of DFS and the high-speed I/O performance of HPSS. Because DFS and HPSS have separate customer bases, with a relatively small overlap, the cost of providing the high-speed I/O through DFS was deemed too high. Therefore, at the fileset level, two data management configuration options, corresponding to the HSM integration philosophies previously mentioned, are provided:

- HPSS is used strictly as an archive facility for DFS, making the integrated DFS/HPSS system a traditional HSM. Access to the name and data space is provided only through the DFS interfaces. Filesets managed with this option are called *archived* filesets.
- Consistency between HPSS and DFS name and data spaces is maintained. DFS data is archived to HPSS, but access to the name and data space is available through both DFS and HPSS interfaces. Updates made through the DFS interface are visible through the HPSS interface and vice versa. Thus, a user may access data through DFS, at standard DFS rates, and when high performance I/O rates are important, use the HPSS interface. Filesets managed with this option are called *mirrored* filesets.

Architectural Overview

The design for integrating HPSS with the Episode file system is shown in Figure 1. The details for each component are discussed in the following sections.

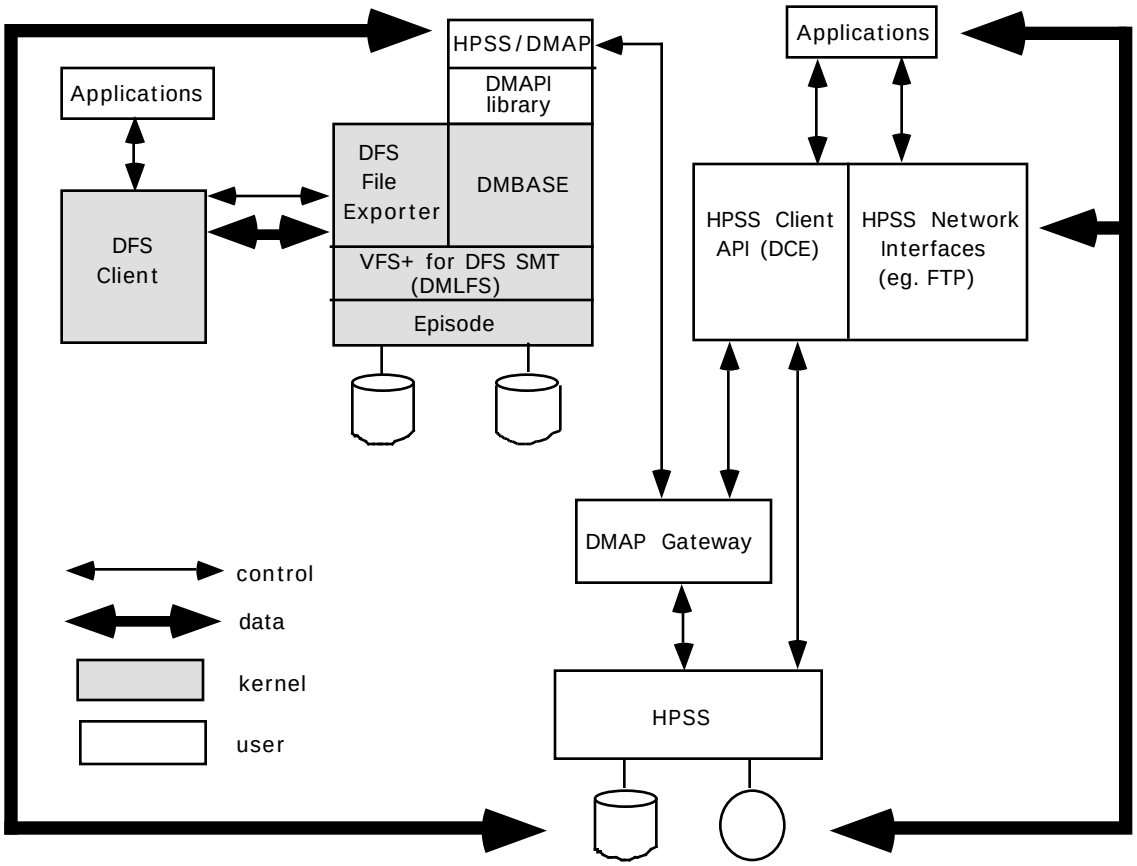

Figure 1. DFS/HPSS DMAPI Integration Architecture

DMAPI Implementation for DFS

The DMAPI implementation for DFS, called the DFS Storage Management Toolkit (DFS SMT), is fully compliant with the corresponding standard XDSM specification. In addition, it provides the following optional features; persistent opaque DM attributes, persistent event masks, persistent managed regions, non-blocking lock upgrades and the ability to scan for objects with a particular attribute.

The bulk of DFS SMT is implemented in the file server. DFS SMT has both user and kernel space components. A user space shared library implements all APIs in the DMAPI specification. A device driver provides communication between the user space component of DFS SMT and the kernel component.

The DFS SMT kernel component consists of two sub-components: a file system independent layer (DMBASE) and a file system dependent layer (DMLFS). DMBASE receives requests via the device driver, processes them and responds to the DMAP via the driver. It maintains DMAPI sessions, DMAPI tokens, event queues, and the registered disposition of events for various file systems. This layer is also responsible for receiving events from the DMLFS and dispatching them to the DMAP. For each DMAPI call, DMBASE is responsible for making appropriate calls to the managed file system via DMLFS.

DFS uses an enhanced virtual file system[14], VFS+, that is implemented for the DFS client and any physical file systems exported by DFS servers. VFS+ was augmented to fetch and store DM attributes, provide persistent managed regions and events, perform invisible I/O, purge data from files, and verify file residency.

DMLFS was designed as a layer in the stackable virtual file system model[15] and is an implementation of the VFS+ layer for DFS SMT. DMLFS operations are responsible for generating events. It consults with DMBASE to determine if any DMAP has registered to receive notification for events related to that particular operation and then generates the events. If the event is synchronous, it causes the file system operation to wait for a response to the event before proceeding. It also provides for interlocking between DFS SMT requests and file system calls.

To support persistent DM-related metadata, an extended attribute facility was provided in Episode. DM attributes, event masks, managed regions, and attribute change times (dtime values) are stored as extended attributes. This eliminates the need to store the attributes in a separate auxiliary file visible in the file name space. These extended attributes are treated as file metadata and changes to attributes are logged.

Episode was modified to support files that become sparse by punching holes that release disk resources. With a conventional sparse file, reading from a hole returns zeroes. To assume these same semantics for a hole that exists because the DMAP migrated the data to tertiary storage is incorrect. In this case, the DMAP must retrieve the data from tertiary storage. Hence, a facility is provided in Episode to mark file blocks as being off-line (in tertiary store) instead of as a hole. This allows the file server to handle partially resident files.

To prevent blocking file server (kernel) threads while waiting for a response to an event, a mechanism to notify the client to retry after a specified interval of time was added. The retry interval is exponentially backed off on each retry.

The DFS fileset dump and restore capability was augmented to include extended attributes and migrated regions. Migrated data is not recalled when a dump is taken, producing an "abbreviated" dump. Checks were added to DFS and retrofitted into prior releases to prevent restoring an abbreviated dump with a file system that may not be able to interpret its contents.

HPSS Implementation for DFS Support

The HPSS collaboration developed two new components, an HPSS Data Management Application (DMAP) and a DMAP Gateway. The HPSS Name Server, Bitfile Server, and Client API also required modifications.

HPSS/DMAP

HPSS Data Management Application (HPSS/DMAP) is responsible for initiating and coordinating file and data interactions between Episode and HPSS. It catches and processes desired name and data space events generated by Episode sent through the DFS SMT; migrates file data from Episode to HPSS; purges unneeded Episode file data after that data migrates to HPSS; and processes requests originating in HPSS interfaces that require either Episode name or data resources. HPSS/DMAP resides on the same machine as the disk(s) managed by the Episode file system.

For portability, HPSS/DMAP communicates with HPSS components via XDR over TCP sockets. The HPSS/DMAP also exists to isolate dependencies on DFS, Episode and DCE.

HPSS/DMAP registers to receive name and data space events. After catching a name space event involving a mirrored fileset, the appropriate requests are made to HPSS to keep the name spaces synchronized. For archived filesets, only create and destroy name space events are processed, but no HPSS resources are utilized. HPSS/DMAP receives data space events when a file is read, written, or truncated and the data region involved is registered with the DFS SMT. HPSS/DMAP is responsible for caching data to Episode that is not present; invalidating HPSS data, if necessary; and manipulating data regions to minimize the occurrence of events involving the same region. HPSS/DMAP can cache partial files.

HPSS/DMAP provides an interface for HPSS to request that an action occur in DFS to keep the HPSS and DFS name and data spaces synchronized. This mechanism is only used with mirrored filesets and occurs when an HPSS client requests to create, delete or modify a name space object. This interface is also used by HPSS to migrate or purge data from Episode disks. Before file data can be altered through HPSS interfaces, the data must first be purged from Episode disks. The capability to forward DCE credentials is provided, enabling HPSS/DMAP to make DFS requests on behalf of the user.

HPSS/DMAP migrates file data from Episode to HPSS. To free Episode disk resources, it also purges file data from Episode. Only the data that has been modified on Episode is migrated to HPSS, thus, a minor modification to a very large file will not result in re-migrating the entire file. Because policies for migrating and purging data are separately configurable, file data migrated from Episode is not automatically purged. In many cases, data for a given file is present in both Episode and HPSS and that data can be read from either interface without any data movement between Episode and HPSS.

DMAP Gateway

The DMAP Gateway is a conduit between HPSS/DMAP and HPSS. HPSS servers use DCE/RPCs, the Encina transaction manager, and Transarc's SFS for metadata managing HPSS objects. The DMAP Gateway encodes requests using XDR and sends them via sockets to HPSS/DMAP and translates XDR from the HPSS/DMAP to DCE/TRPC/Encina calls to the appropriate HPSS server. XDR and sockets were used for portability. When a connection between the HPSS/DMAP and Gateway is made, mutual authentication occurs.

The DMAP Gateway keeps track of the location in DFS and in HPSS of all the filesets it manages. For scalability, multiple DMAP Gateways are supported. However, a given DMAP Gateway will only operate on the filesets it manages. At this time, only filesets managed by DFS and HPSS are supported.

In addition to translating requests between HPSS/DMAP and HPSS servers, the DMAP Gateway keeps fileset request statistics and internal DMAP Gateway resource utilization. Heavily used filesets can be identified and management of these filesets could be distributed across multiple DMAP Gateways to improve performance.

Name Server

Support for filesets (disjoint directory trees) was added to the Name Server. HPSS supports three types of filesets:

- HPSS-only filesets contain objects accessible only through standard mechanisms available to HPSS Name Server clients.
- Archived filesets are directory structures with private files containing copies of file data from any DFS files that have migrated to HPSS. Path names to directory objects are generated by HPSS/DMAP based on configuration policies. At any given time, the data in these files may be out of date with the DFS data. Access to objects in these filesets is restricted to the DMAP Gateway and HPSS root.
- Mirrored filesets contain objects that are kept synchronized between HPSS and DFS. Objects in mirrored filesets have corresponding objects in DFS and HPSS with identical names and attributes. HPSS clients access these objects as before, but whenever a client's request alters the name space, the alteration occurs as a side effect of the change made in Episode. Such requests are forwarded by the Client API to the DMAP Gateway and then to HPSS/DMAP. HPSS/DMAP makes appropriate Episode system calls, causing events to be generated. Processing the event ultimately results in a request to the HPSS Name Server to perform the appropriate action. Requests processed this way include create, remove, symlink, hardlink, and permission changes. This mechanism is transparent to the HPSS client.

Bitfile Server

The Bitfile Server was modified to recognize when file data on HPSS was inconsistent with its DFS counterpart. Inconsistent data occurs whenever file data altered through the DFS interface has not yet migrated to HPSS. The Bitfile Server was modified to return a special error value whenever an HPSS I/O request involves file data on a mirrored fileset that is in this state. This error signals the Client API to request that the file data be migrated to the HPSS file before retrying the I/O request. Before the Client API reissues any write requests, it first requests that the Episode data be invalidated.

HPSS Client API

The HPSS client API was modified to handle fileset domains, recognize an object's fileset type, determine which server to contact to process a request for a given object, and cross junctions (fileset domains). Junctions provide the ability to link filesets managed by the same Name Server or by another HPSS Name Server. With this mechanism multiple HPSS name spaces can be joined to form a single name space.

For HPSS-only filesets, the Client API issues the same requests as in prior HPSS releases. Behavior of objects in this type of fileset is identical to current releases of HPSS, unless the client crosses a junction. Access to objects in this type of fileset is only allowed through non-DFS HPSS interfaces. If a system administrator decides to also permit access through DFS, then tools must be run to import the HPSS metadata into the Episode file system.

For archived filesets, client access through HPSS interfaces are prohibited. In general, file data in this type of fileset must be accessed through DFS, but in special cases, a system administrator may access the HPSS objects.

If an HPSS client request will alter the name space of a mirrored fileset, the Client API forwards the request to the DMAP Gateway. The Gateway sends the appropriate request to HPSS/DMAP which then requests Episode to make the modification. When Episode processes the request, a DMAPI event is generated and the HPSS/DMAP event handler makes the necessary HPSS calls to keep the name spaces synchronized. All name space modifications in mirrored filesets are the result of DMAPI events generated either by DFS calls, or by Client API calls forwarded to the HPSS/DMAP.

The Client API was modified to provide a consistent view of file data between DFS and HPSS interfaces for mirrored filesets. The Client API recognizes the new error returned by the Bitfile Server when data is inaccessible through HPSS because of DFS activity. When this error occurs, the Client API initiates steps to migrate and purge Episode data so the HPSS and Episode data remain synchronized.

DMAPI Extensions

To support mirrored filesets our design requires a superset of the DMAPI standard. DFS SMT was extended to support synchronous post events generated after Episode completes a name space modification but before the original user request completes. These events are necessary to keep HPSS and DFS name spaces synchronized. Delivery of synchronous post events is guaranteed to be fast. Also, to support mirrored filesets, the ability to register and generate events for permission changes to a file or directory were added. These events occur whenever the owner, group, mode, or ACL is changed.

Unique issues are raised because DFS supports filesets and aggregates, while the DMAPI was designed for traditional file systems. These issues are addressed by the DFS SMT. Specifically, to handle fileset destruction additional name space events were defined. Mounting and unmounting aggregates is treated by DMAPI as a single event. The actual implementation for DFS SMT is a multi-step process, with complex recovery logic, because events must be generated and processed for each fileset on the aggregate.

Additional fields were added to DMAPI structures. A new field was added to the events structure to associate paired events (pre and post events). A new field was added to the region structure to store user defined opaque data. Two fields were added to the statistics structure that identify the presence of ACLs associated with a file and to store a fileset identifier.

Additional DMAPI management interfaces were added to handle ACLs, DCE security authentication information, to enumerate information about filesets, and to determine the handle for a named file in a known directory.

Examples

To illustrate the interaction between the system components, an overview of creating and reading a file follows.

DFS Create Example

When a client requests to create a file through DFS, DFS SMT generates a sequence of DMAPI create events. HPSS/DMAP catches these events and if the fileset type is mirrored it issues a request to the DMAP Gateway to create the HPSS file. The DMAP Gateway issues a request to the Name Sever and Bitfile Server through the Client API to create a name space object and a bitfile. After these HPSS resources have been created, the DMAP Gateway responds to HPSS/DMAP, which then responds to the DMAPI create events. Finally, the DFS file resources are allocated and after that completes, the DFS client's create request completes.

When a client requests to create a file in an archived fileset, DFS SMT still generates create events. However, HPSS/DMAP does not immediately create the HPSS file. It simply marks the file as a candidate for migration and responds to the create event. HPSS resources will be allocated when the file is migrated to HPSS. Since HPSS resource

allocation is delayed, the DFS client's request completes only slightly slower than a create request in a DFS fileset not integrated with HPSS.

HPSS Read Example

This example applies only to mirrored filesets. When an HPSS client requests to read data from a file, the request is issued through the Client API to the Bitfile Server. If the Bitfile Server has the desired data and that data is consistent with the data mirrored on DFS, the Bitfile Server proceeds to read the data. However, if the data is inconsistent with DFS, the Bitfile Server returns an error to the Client API.

Upon receiving an inconsistent data error, the Client API requests the DMAP Gateway managing the fileset to migrate the data from Episode to HPSS. The DMAP Gateway contacts the HPSS/DMAP, which uses DFS SMT to migrate the data from Episode to HPSS, and then marks the data as synchronized. HPSS/DMAP responds to the DMAP Gateway, which responds to the Client API. At that point, the desired data has migrated to HPSS and the Client API retries the read request.

Performance Comparisons

Performance data was gathered from tests run on two platforms; an AIX 4.1 system and a Solaris 2.5.1 system. The performance of these systems varied, due to different processor speeds, but the overall trends were identical. We measured performance for a DFS only fileset and compared it to archived and mirrored filesets. We also measured the performance for an HPSS only fileset and compared it to a mirrored fileset. To determine the impact of the DFS SMT on file system performance, we measured performance on Episode aggregates with and without DFS SMT. In both cases, the DM application (HPSS/DMAP) was not running. The numbers reported are averages of several runs, where each run was performed with an empty cache.

DFS SMT Performance

Sequential read/write performance was measured using the UNIX "cp" command to copy a file between the Episode aggregate and a UFS aggregate. Our tests found that an aggregate configured with the DFS SMT layer enabled to support DM applications, had a 1.5-2.5% decrease in read/write performance. The small overhead due to the extra layer in the file system is overshadowed by the time taken for synchronous disk I/O.

In addition to read/write performance measurements, we also ran the NFS Connectathon benchmark suite that attempts to thoroughly exercise file system functionality such as file and directory creation and deletion, lookup, setattr, getattr, chmod, stat, read, write, link, rename etc. We found that it takes 5.7% longer for file system calls to execute on a DFS SMT managed aggregate. In contrast to the read/write tests, the Connectathon test may lead to a significantly smaller proportion of synchronous I/O operations because of cached metadata. Without the masking effect of I/O, the overhead incurred by the DFS SMT layer is more perceptible.

We are working towards making the DFS SMT code more efficient, and expect the overhead to decrease in future releases.

DFS Performance

Archived filesets generate and process events whenever files are created and deleted, incurring some overhead. Events are not posted for any other name space events, such as

link, rename, etc., so these operations, perform at the same rate as a DFS only fileset. Unless file data must be staged from HPSS, I/O performance rates for an archived fileset is identical to those of a DFS only fileset.

Creating a file in an archived fileset incurs an overhead between 30-40% above creates in a DFS only fileset. This overhead is attributed to event processing in DFS SMT and to event handling by HPSS/DMAP which must set DM attributes, which involves I/O. The cost incurred for processing file deletes has some variance. At a minimum, the DM attributes must be checked to determine if the file has migrated to HPSS. In this case, the overhead for deletes is 20% greater than deletes in a DFS only fileset. If the file has migrated to HPSS, HPSS/DMAP must log object handle information for the file and the overhead increases by the amount of time it takes to write to the log.

The overhead associated with client I/O to files is insignificant for both mirrored and archived filesets, unless the data must be cached to DFS. Moving data between DFS and HPSS is highly dependent on disk speed, network speed, and system load, but during our tests, data transfers between DFS and HPSS were as fast as permitted by the hardware.

HPSS Performance

Mirrored filesets incur an additional overhead for any name space activity that alters the name space. Such activity includes creates, unlinks, renames, and owner or permissions changes. Name space activity that does not alter the name space, such as “ls” and “cd”, perform at the same rate as HPSS only filesets. In general, the overhead for keeping the DFS and HPSS name spaces synchronized is about 10% greater than the same activity in an HPSS only fileset. The overhead seen from HPSS is significantly lower than the overhead seen from DFS because HPSS has a higher cost for name space updates than DFS. This cost is largely due to the infrastructure and efforts are underway to reduce it.

I/O throughput for mirrored filesets is the same as for HPSS-only filesets, if all the required data is present on HPSS media. If data must be migrated or purged from Episode, HPSS I/O may be delayed. The length of the delay depends on the amount of data to be migrated, network and media speed, and the load currently on the DFS and HPSS systems.

Data and name space activity on HPSS-only filesets perform at the same rates as previous HPSS releases. The cost incurred by the use of filesets is insignificant.

Conclusions

The integrated DFS/HPSS system is flexible enough to support a variety of environments. Data I/O rates for both DFS and HPSS are consistent with the rates before DMAPI was implemented, except when data must be migrated or staged. Supporting partial data migration and caching allows pieces of files to be moved, improving the speed of migrating and caching file data between DFS and HPSS.

Providing consistent name spaces between DFS and HPSS is possible, but at an additional cost. However, the additional cost of creating files becomes less significant as the files increase in size.

Drawbacks of the integration include greater administrative complexity and performance overhead arising from DMAPI, especially when there is a need to keep the HPSS and DFS name spaces synchronized.

Future Work

Future DFS work includes support for cloning filesets on DFS SMT managed aggregates, to enable fileset movement and replication. When dumping a fileset, DFS will allow for purged data to be recalled from tertiary storage, supporting full and abbreviated fileset dumps. Extensions will be added to the DFS client-to-file-server protocol, allowing the DM attributes to be examined.

HPSS/DMAP can be ported to other platforms that support DMAPI, allowing these file systems to be archived in HPSS, however, mirroring name spaces will be impossible without DMAPI extensions. Functionality must be added to HPSS/DMAP to support cloning and replication.

Though we have made DFS/Episode conform to DMAPI, it is not a perfect match since DMAPI mainly addresses local file systems like UFS. To provide a better match between DFS and DMAPI, we would like the following extensions to DMAPI:

- To support backup applications, it must be possible to capture all attributes for a file (ACL, Episode file property lists, etc.).
- DFS allows filesets to move between file servers at different sites. We would like to move filesets without recalling all the data. The DMAP at the new site should be able to interpret DM attributes correctly and recall migrated data. This requires DM attributes to contain location information.

HPSS would like the following extensions to DMAPI:

- Support for permission change events and additional events for name space synchronization.
- Modification to the locking and I/O interfaces for better parallel file system support.

References

[1] S. S. Coleman, R. W. Watson, R. A. Coyne, and H. Hulen, “The Emerging Storage Management Paradigm”, *Proceedings Twelfth IEEE Symposium on Mass Storage Systems*, Monterey, CA, April, 1993.

[2] Kazar et al., “DEcorum File System Architectural Overview,. *of USENIX Summer Conference*, Anaheim, California, 1990.

[3] D. Teaff, R. W. Watson and R. A. Coyne, "The Architecture of the High Performance Storage System (HPSS)", *Proceedings Goddard Conference on Mass Storage & Technologies*, College Park, MD, Mar. 1995.

[4] CAE Specification: Data Storage Management - A Systems Approach, The Open Group, 1997.

[5] Cray Research, Inc., Data Migration, UNICOSä System Administration Guide Volume 1 (SG-2113), 1991.

[6] M. J. Teller and P. G. Rutherford , Petabyte File Systems Based on Tertiary Storage, htpp:/www.emass.com/World_HQ/Collaterals/Whitepapers/Petabyte.pdf.

[7] R. E. Bredehoft, “Advances in E-Systems Modular Automated Storage System (EMASSä) For the Cray Environment”, *Cray User Group Spring Proceedings*, Berlin, April 1992.

[8] M. W. Collins. and C. W. Mexal, “The Los Alamos Common File System,” *Tutorial Notes*, Ninth IEEE Symposium on Mass Storage Systems, October 31, 1988.

[9] F. McClain, “Distributed Computing Solutions, DataTreeä and UniTreeä : Software for File and Storage Management,” *Digest of Papers Tenth IEEE Symposium on Mass Storage Systems*, 1990.

[10] R. A. Coyne and R. W. Watson, "The Parallel I/O Architecture of the High Performance Storage System (HPSS)", *Proceedings Fourteenth IEEE Symposium on Mass Storage Systems*, Monterey, CA, Sept. 1995.

[11] R. Hyer, R. Ruef, and R. W. Watson, "High Performance Direct Network Data Transfers at the National Storage Laboratory*," Proceedings Twelfth IEEE Symposium on Mass Storage*, Monterey, CA, Apr. 1993.

[12] Chutani et al., “The Episode File System”, *Proceedings of the Winter USENIX Conference*, San Francisco, California, 1992.

[13] Open Group, "Systems Management: Data Storage Management (XDSM) API", *Open Group CAE Specification*, Prentice-Hall, March 1997.

[14] Kleiman, "Vnodes: An Architecture for Multiple File System Types in SUN UNIX", *Proceedings of the USENIX Summer Conference*, 1986.

[15] J. Heidemann and G. Popek, "File System Development with Stackable Layers", *ACM Transactions on Computer Systems*, 12/1, February 1994.